CSE331 Introduction to Algorithm
Lecture 4: Solving Recurrences

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 11, 2017

Introduction

- Reference: Sections 4.3, 4.4 and 4.5 of the textbook Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.
- In Lecture 3, the running times were given by recurrence relations.

$$\text{Merge Sort: } T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad (1)$$

$$\text{Binary Search: } T(n) = T(\lfloor n/2 \rfloor) + \Theta(1) \quad (2)$$

- It is often going to be the case.
  - Many algorithms are recursive.
  - In particular, divide and conquer algorithms.
- In this Lecture, we will see how to solves some recurrence relations that often arise when analyzing algorithms.
- The function $T(n)$ in this lecture denotes a running time on input of size n, and thus $T(n) > 0$ for all integer $n \geqslant 1$.

The Substitution Method

The substitution method

The *substitution method* for solving recurrences comprises two steps.

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show that the solution works.

- Example. We want to solve a recurrence similar to (1):

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \tag{3}$$

- More precisely, we want to find a good upper bound on $T(n)$.
- We first guess that $T(n) = O(n \log n)$.

The Substitution Method

- So we want to prove by mathematical induction that $T(n) \leqslant cn \log n$ for some constant c.
- Recall that a proof by induction requires two steps:
  - The *base step*, and
  - the *inductive step*.
- We begin with the inductive step.

Inductive Step

- So we assume that $n > 1$ and $T(m) \leqslant cm \log m$ for all $m < n$, and we want to prove that $T(n) \leqslant cn \log n$.

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leqslant 2c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor) + n \\ &\leqslant 2c(n/2)\log(n/2) + n \\ &= cn(\log(n) - \log 2) + n \\ &= cn(\log(n) - 1) + n \\ &= cn \log n + (1 - c)n \end{aligned}$$

- So if we choose $c \geqslant 1$, we have $T(n) \leqslant cn \log n$, which completes the inductive step.

Base Step

- Base step: We would like to prove that $T(n) \leqslant cn \log n$ for $n = 1$.
- Problem:
  - Since $\log 1 = 0$, it means $T(1) \leqslant 0$.
  - $T(1)$ should be positive as it is a running time.
- Solution:
  - We will use $T(2)$ and $T(3)$ as base cases, because for any $n > 3$, the recurrence goes through $n = 2$ or $n = 3$ before reaching $n = 1$.
  - We want to have

$$T(2) \leqslant 2c \log 2 = 2c$$
$$T(3) \leqslant 3c \log 3$$
$$1 \leqslant c \quad \text{(from previous slide)}$$

  - So we choose

$$c = \max\left(1, \frac{T(2)}{2}, \frac{T(3)}{3 \log 3}\right)$$

The Substitution Method

- In summary, we have proved the following.
- Let c be the constant

$$c = \max\left(1, \frac{T(2)}{2}, \frac{T(3)}{3\log 3}\right).$$

- Let $P(n)$ be the property $T(n) \leqslant cn\log n$.
- Then we have proved that $P(2)$ and $P(3)$ are true, and the calculations on Slide 6 show that

$$P(\lfloor n/2 \rfloor) \text{ is true implies } P(n) \text{ is true.} \tag{4}$$

- As $P(2)$ is true, Property (4) implies that $P(4)$ and $P(5)$ are true.
- So $P(2), P(3), P(4)$ and $P(5)$ are true, which implies that $P(m)$ is true for all $2 \leqslant m \leqslant 11$.
- ...

The Substitution Method

- We have proved by induction that for all $n \geqslant 2$,

$$T(n) \leqslant cn \log n$$

  where the constant c is given by

$$c = \max\left(1, \frac{T(2)}{2}, \frac{T(3)}{3 \log 3}\right).$$

- It means that Equation (3) solves to $T(n) = O(n \log n)$.
  - Here the n_0 from the definition of the $O(\cdot)$ notation is 2.

The Substitution Method: Example 2

- We now want to find an upper bound for

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \tag{5}$$

- We guess that $T(n) = O(n)$.
- So we try to prove that $T(n) \leqslant cn$ for some constant c.
- The inductive step would be:

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leqslant c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + 1 \\ &= cn + 1 \end{aligned}$$

- This approach failed.

The Substitution Method: Example 2

- Previous attempt was off by 1 only.
- So we make a small change: We try to prove that $T(n) \leqslant cn + d$ for some constants c, d.
- Inductive step:

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leqslant c(\lceil n/2 \rceil) + d + c(\lfloor n/2 \rfloor) + d + 1 \\ &= cn + 2d + 1 \end{aligned}$$

- So it works if $2d + 1 \leqslant d$, which means $d \leqslant -1$.
- We choose $d = -1$.
- As we can choose $c > 0$ freely, we can handle the base cases.
- We just proved that $T(n) \leqslant cn - 1$ for some constant $c > 0$, and thus $T(n) = O(n)$.

Avoiding Pitfalls

$$
\begin{aligned}
T(n) &\leqslant 2c\lfloor n/2 \rfloor + n \\
&\leqslant cn + n \\
&= O(n) \qquad\qquad \leftarrow \textit{wrong!}
\end{aligned}
$$

- Problem: In the inductive step, we should prove the *exact form* of the inductive hypothesis.

The Recursion Tree Method

- The substitution method often gives short proofs.
- Difficulty: guessing the solution.
- We can use the *recursion tree* method to make a good guess.
  - We already saw this method in Lecture 2.
- So one approach to solve recurrences is:
  1. Guess the solution using the recursion tree method.
     - ★ We can afford to be sloppy, as the goal is to make a guess.
  2. Prove it using the substitution method.
     - ★ Needs to be rigorous (see Slide 12).

The Recursion Tree Method: Example 1

- We want to guess a good upper bound for

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \tag{6}$$

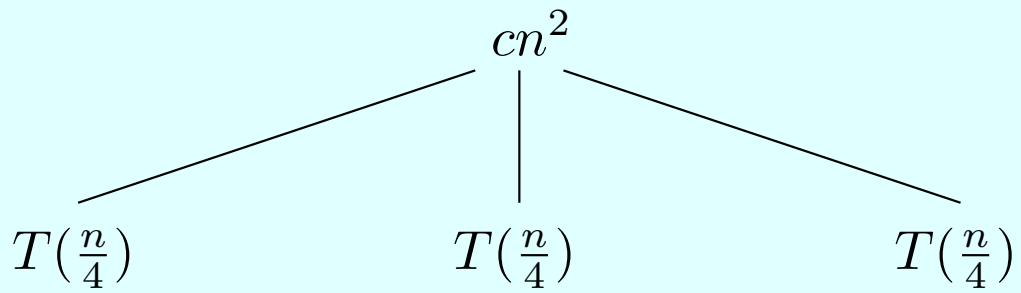
- We can afford to be sloppy. So we assume that n is a power of 4.
- It means that we can remove the floor functions.
- So we rewrite Equation 6:

$$T(n) = 3T(n/4) + cn^2 \qquad \text{for some constant } c.$$

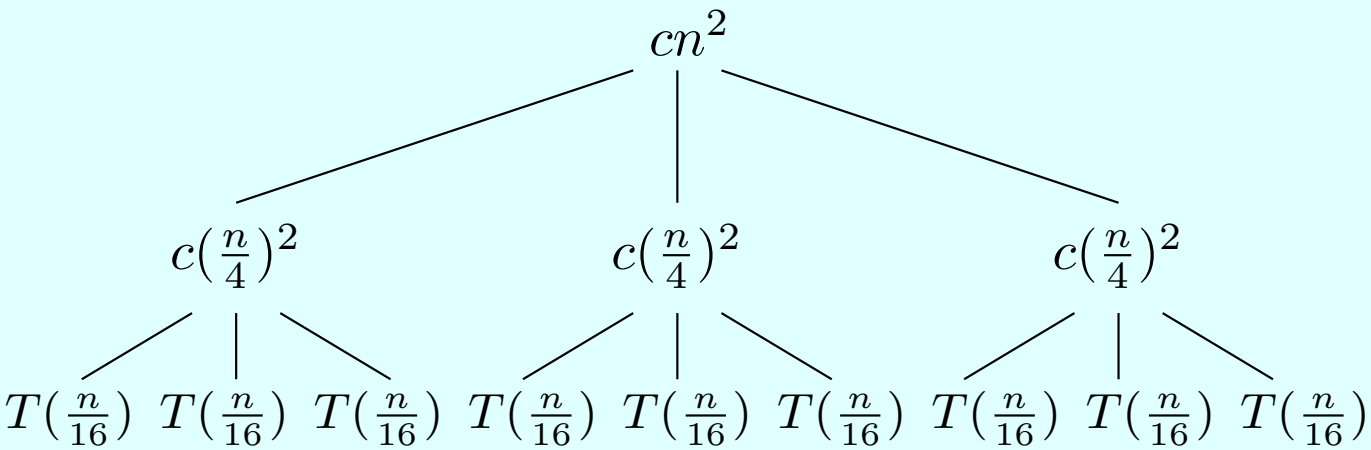
The Recursion Tree Method: Example 1

$$T(n)$$

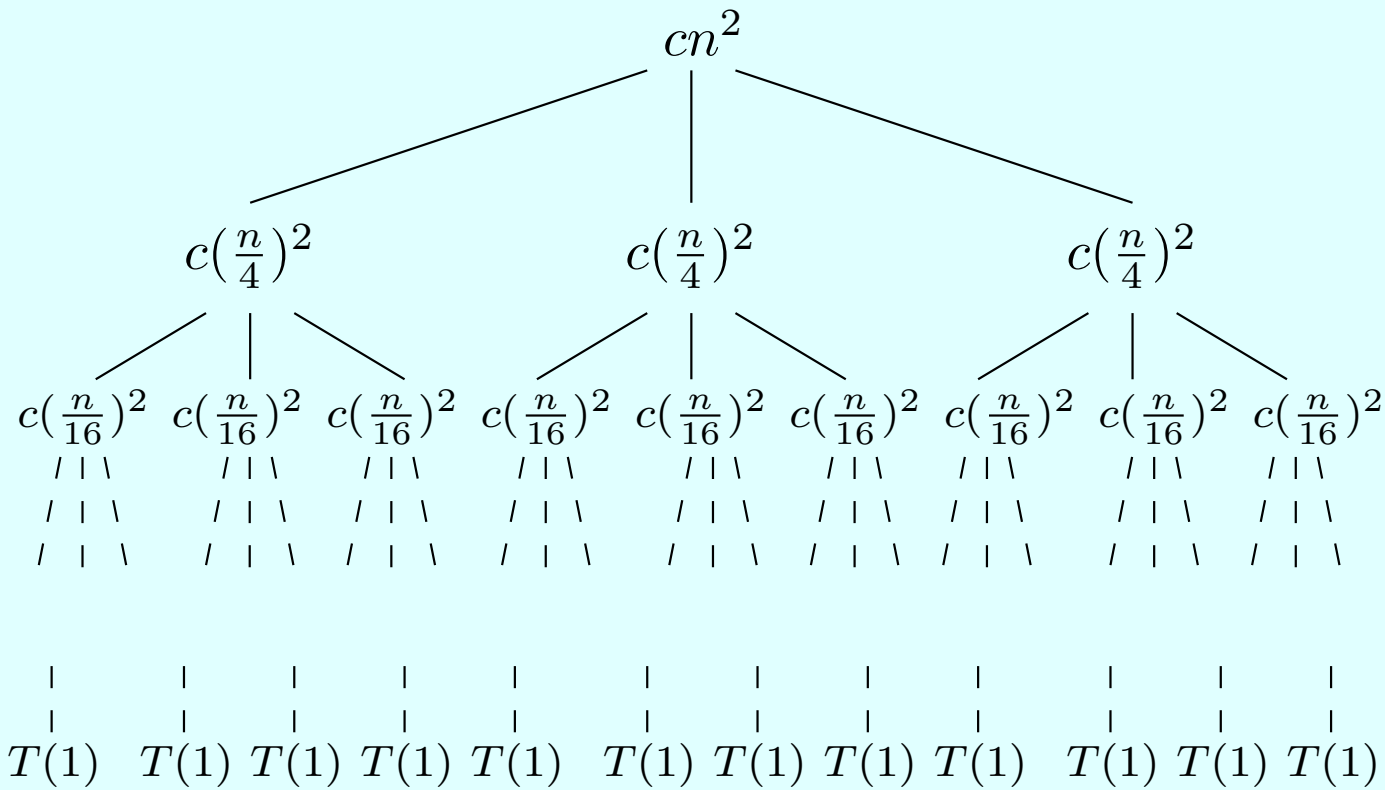
The Recursion Tree Method: Example 1



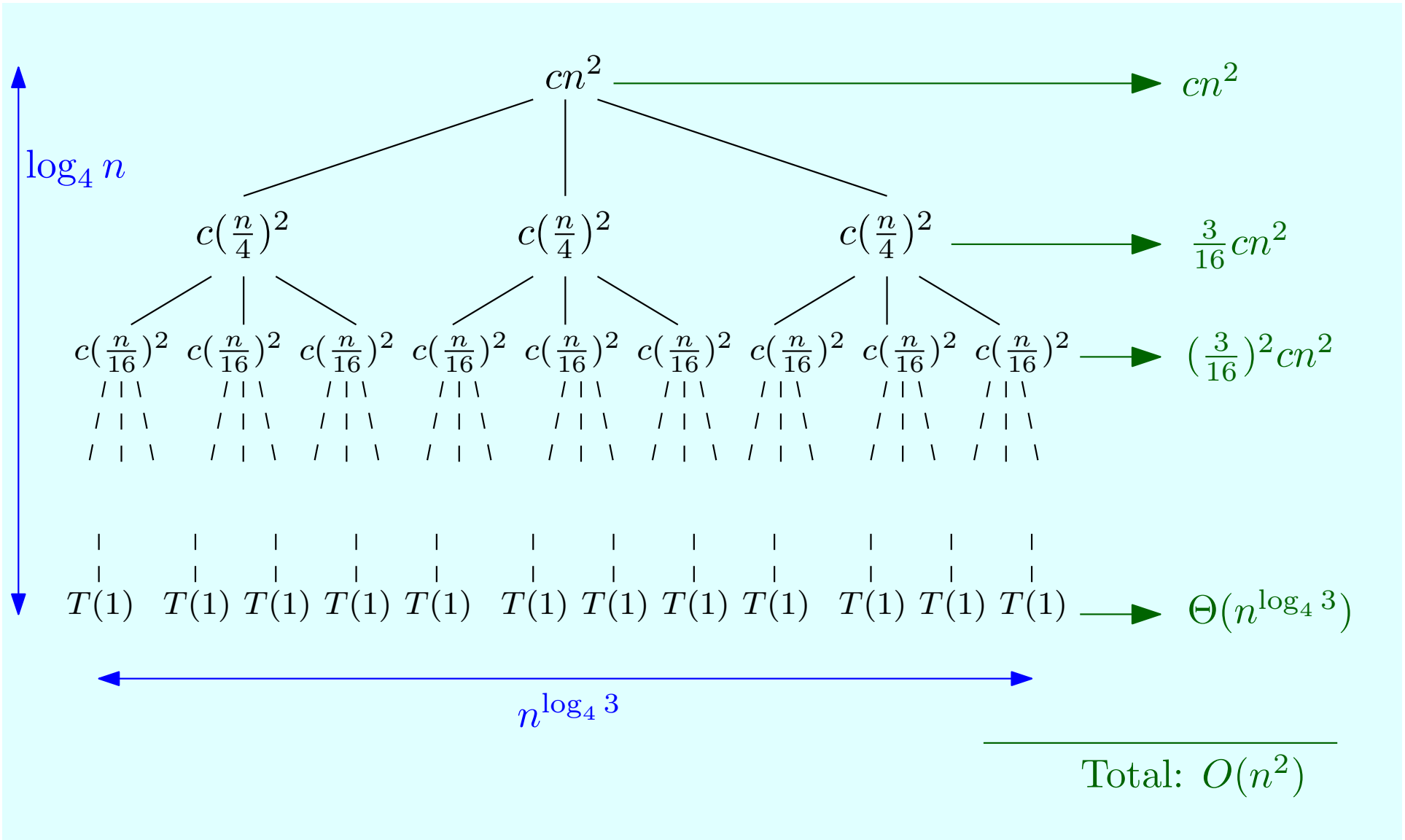
The Recursion Tree Method: Example 1



The Recursion Tree Method: Example 1



The Recursion Tree Method: Example 1



The Recursion Tree Method: Example 1

- Let h denote the height of the tree.
- The size of the subproblems decreases by a factor 4 at each level.
- So $4^h = n$, which means that $h = \log_4 n$.
- So the number of leaves is

$$3^h = 3^{\log_4 n} = 3^{\frac{\log_3 n}{\log_3 4}} = 3^{(\log_3 n)(\log_4 3)} = n^{\log_4 3} \simeq n^{0.792}$$

- At depth $i < h$:
  - The number of nodes is 3^i.
  - The size of each subproblem is $n/4^i$.
  - The total cost over all nodes at depth i is

$$3^i c(n/4^i)^2 = c\left(\frac{3}{16}\right)^i n^2.$$

The Recursion Tree Method: Example 1

- At the leaves, the total cost is $\Theta(n^{\log_4 3})$ as there are $n^{\log_4 3}$ leaves and each leaf has cost $\Theta(1)$.
- Overall

$$\begin{aligned} T(n) &= O(n^{\log_4 3}) + cn^2 \sum_{i=0}^{h-1} \left(\frac{3}{16}\right)^i \\ &\leqslant O(n^{\log_4 3}) + cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i \\ &= O(n^{\log_4 3}) + cn^2 \frac{16}{13} \\ &= O(n^2) \end{aligned}$$

so $T(n) = O(n^2)$.

The Recursion Tree Method: Example 1

- We have just guessed that Equation (6) yields $T(n) = O(n^2)$.
- We now *prove* it with the substitution method.
- We rewrite Equation (6) using an unknown constant c:

$$T(n) \leqslant 3T(\lfloor n/4 \rfloor) + cn^2.$$

- Now we prove by induction that $T(n) \leqslant dn^2$ for some constant d.

$$\begin{aligned} T(n) &\leqslant 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leqslant 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leqslant 3d(n/4)^2 + cn^2 \\ &= \left(\frac{3}{16}d + c\right) n^2 \end{aligned}$$

The Recursion Tree Method: Example 1

- So if we choose $d \geqslant 16c/13$, we have proved that $T(n) \leqslant dn^2 = O(n^2)$.
- We also have $T(n) = \Omega(n^2)$. Why?
  - It follows directly from Equation (6) that $T(n) \geqslant \Theta(n^2)$.
- So we proved the tight bound $T(n) = \Theta(n^2)$.

The Recursion Tree Method: Example 2

Problem

Find a good upper bound for $T(n) = T(n/3) + T(2n/3) + O(n)$.

(See textbook p. 91)

The Master Method

Theorem (Master Theorem)

Let $a \geqslant 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n) > 0$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. *If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.*
2. *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.*
3. *If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leqslant cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

The Master Method

- I will not prove the master theorem in CSE331.
- But here is some intuition.
- **Case 2:** $f(n) = \Theta(n^{\log_b a})$.
  - A simple calculation shows that for each level of the recursion tree, the cost is $\Theta(n^{\log_b a})$.
  - As $b > 1$, the height of the recursion tree is $\Theta(\log n)$.
  - So the running time is $T(n) = \Theta(n^{\log_b a} \log n)$.
- **Case 1:** $f(n)$ is much smaller than $n^{\log_b a}$.
  - As there are $n^{\log_b a}$ leaves, the cost of the leaves is $\Theta(n^{\log_b a})$.
  - So the leaves dominate the running time, and $T(n) = \Theta(n^{\log_b a})$.
- **Case 3:** $f(n)$ is much larger than $n^{\log_b a}$.
  - In this case the cost is dominated by the root node of the tree.
  - So $T(n) = \Theta(f(n))$.

The Master Method: Example 1

- The running time of *binary search* is given by:

$$T(n) = T(\lfloor n/2 \rfloor) + 1$$

- So $a = 1$, $b = 2$ and $f(n) = 1$.
- $n^{\log_b a} = n^0 = 1$.
- As $f(n) = 1 = \Theta(n^{\log_b a})$, we are in case 2, and thus

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n).$$

The Master Method: Example 2

- The running time of *merge sort* is given by:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n).$$

- The floor and ceiling function can be discarded when we apply the master theorem, so we rewrite it:

$$T(n) = 2T(n/2) + \Theta(n).$$

- Thus $a = 2$, $b = 2$ and $f(n) = \Theta(n)$.
- $n^{\log_b a} = n$.
- We are in case 2, and thus

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n).$$

The Master Method: Example 3

$$T(n) = 9T(n/3) + n$$

- We have $a = 9$, $b = 3$ and $f(n) = n$.
- So $n^{\log_b a} = n^{\log_3 9} = n^2$.
- Therefore $f(n) = n = O(n^{\log_b a - \varepsilon})$ for $\varepsilon = 1$.
- We are in Case 1 of the master theorem, and thus $f(n) = \Theta(n^2)$.

The Master Method: Example 4

- Consider the recurrence relation $T(n) = T(2n/3) + 1$
- $a = 1$, $b = 3/2$ and $f(n) = 1$.
- $n^{\log_b a} = n^0 = 1$.
- We are in Case 2, and

$$T(n) = \Theta(\log n).$$

- What is the connection with binary search?

The Master Method: Example 5

- Consider the recurrence relation $T(n) = 3T(n/4) + n \log n$
- $a = 3$, $b = 4$ and $f(n) = n \log n$.
- $n^{\log_b a} \simeq n^{0.8}$.
- $f(n) = \Omega(n^{\log_b a})$, so we are in Case 3, and

$$T(n) = \Theta(n \log n).$$

The Master Method: Example 6

- Consider the recurrence relation $T(n) = 2T(n/2) + n \log n$
- $a = 2$, $b = 2$ and $f(n) = n \log n$.
- $n^{\log_b a} = n$.
- None of the three cases applies, because we don't have $n \log n = \Omega(n^{1+\varepsilon})$ for any $\varepsilon > 0$.
  - Reason: $\log n$ grows more slowly than n^{ε} for every $\varepsilon > 0$.

The Master Method

- The master theorem provides a general method for solving recurrences.
- The three cases of the master theorem do not cover all possibilities.
  - (See previous slide.)
- You do not need to memorize the master theorem statement. I will give it in the exam.